

Question **1**

Correct

Note de 2,00
sur 2,00

Cette question devrait être faite sans utiliser un compilateur.

Pour chaque case de cette question, indiquez le type qu'il faut mettre devant le nom de variable dans la déclaration pour que l'affectation soit correcte et qu'elle ne change pas le type de la valeur (ex. : `bool x = 2;` est une affectation correcte mais change le 2 d'un entier à un booléen, donc pas accepté comme réponse). Aucune des réponses n'est "const" ni une référence à quelque chose qui pouvait être affecté par copie.
Indiquez 'X' comme type si l'expression ne peut pas être correcte.

Réponse : (régime de pénalités : 0 %)



```
// Soient les déclarations:
struct B { int a; char b; short** c; };
struct A { B** a; B b; string c; A* d; };
struct C { double a; float* b; A** c; };
A* b(int x);
A a;
C c;

// Indiquez les bons types devant les variables suivantes:
```

d = new char;

e = *(b(3));

f = a.d->a[0];

g = &(a.a[0]->b);

h = b(1)->d->b;

i = **(c.c);

j = b(5);


k = c.b[0];

m = a.a->c;

n = a.d->a[0][0];

Réponse bien enregistrée: char*, A, B*, char*, B, A, A*, float, X, B

Ce message vert n'indique pas si la réponse est bonne ou non, seulement que toutes les cases ont été remplies:

Tous les tests ont été réussis ! 

Correct



Question 2

Terminé

Note de 1,60
sur 2,00

Soient les classes

```
1 class PolyBook {  
2 public:  
3     // 2  
4 private:  
5     // 1  
6 }  
7  
8 class Etudiant {  
9     // ...  
10 }
```

Identifier le constructeur et l'attribut de la classe PolyBook.

Si **PolyBook**
est une
agrégation
par
pointeur de
Etudiant,

1. Etudiant *etud_; 2. PolyBook(Etudiant* etud) : etud_(etud) {}

Si **PolyBook**
est composé
directement
de
Etudiant,

1. Etudiant etud_; 2. PolyBook(const Etudiant& etud) : etud_(etud) {}

Si **PolyBook**
est une
agrégation
par
référence
de
Etudiant,

1. Etudiant &etud_; 2. PolyBook(Etudiant& etud) : etud_(etud) {}

Si **PolyBook**
est composé
par
pointeur de
Etudiant,

1. Etudiant *etud_; 2. PolyBook(Etudiant* etud) : etud_(nullptr) { etud_ = new Etudiant (etud); }



Si **PolyBook**
est composé
par
référence
de
Etudiant,

Aucune solution

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 4.

La réponse correcte est :

Si **PolyBook** est une agrégation par pointeur de **Etudiant**,

→ 1. **Etudiant *etud_**; 2. **PolyBook(Etudiant* etud) : etud_(etud) {}**,

Si **PolyBook** est composé directement de **Etudiant**,

→ 1. **Etudiant etud_**; 2. **PolyBook(const Etudiant& etud) : etud_(etud) {}**,

Si **PolyBook** est une agrégation par référence de **Etudiant**,

→ 1. **Etudiant &etud_**; 2. **PolyBook(Etudiant& etud) : etud_(etud) {}**,

Si **PolyBook** est composé par pointeur de **Etudiant**,

→ 1. **Etudiant *etud_**; 2. **PolyBook(const Etudiant& etud) : etud_(nullptr) { etud_ = new Etudiant(etud); }**,

Si **PolyBook** est composé par référence de **Etudiant**,

→ Aucune solution



Question **3**

Correct

Note de 3,00
sur 3,00

Soit la classe Piece ci-dessous. On aimerait pouvoir créer de nouveaux objets à partir d'une pièce déjà existante.

La copie d'un pièce doit respecter ces règles:

- Un objet copié doit avoir son identifiant (id_) incrémenté de 1 par rapport à l'original.
- Un objet copié doit avoir les même fabricants que l'objet de base. Cependant ajouter un fabricant après la copie ne doit impacter que l'objet concerné.
- Tous les autres attributs doivent être copiés tels qu'ils sont.

Vous faites vos définitions à l'extérieur de la classe et vous ne pouvez pas changer les signatures des méthodes ou en ajouter de nouvelles par rapport aux déclarations dans la classe ci-dessous.



```

1 class Piece
2 {
3 public:
4     Piece(const string& nom, const unsigned int id) :
5         nom_(nom), id_(id)
6     {
7         maxFab_ = 10;
8         nombreFab_ = 0;
9
10        fabriquants_ = new string[maxFab_];
11    }
12
13    Piece(const Piece& p);
14
15    ~Piece()
16    {
17        delete[] fabriquants_;
18    }
19
20    void ajouterFab(const string& fab);
21
22    string getNom() { return nom_; }
23    unsigned getId() { return id_; }
24    unsigned getPrix() { return prix_; }
25    unsigned getNombreFab() { return nombreFab_; }
26    unsigned getNombreFabMax() { return maxFab_; }
27    string* getFabriquants() { return fabriquants_; }
28
29 private:
30     string nom_;
31     unsigned id_;
32     double prix_;
33
34     unsigned maxFab_;
35     unsigned nombreFab_;
36
37     string* fabriquants_;
38 };

```

Par exemple:



Test	Résultat
<pre> Piece p6("roue", 0); Piece p7(p6); cout << (p7.getId() == p6.getId() + 1); cout << (p7.getPrix() == p6.getPrix()); cout << (p7.getNom() == p6.getNom()); cout << (p7.getNombreFab() == p6.getNombreFab()); cout << (p7.getNombreFabMax() == p6.getNombreFabMax()); </pre>	11111

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```

1 Piece::Piece(const Piece& p): nom_(p.nom_)
2                               maxFab_(p.maxFab_)
3 {
4     fabriquants_ = new string[maxFab_];
5     for (auto i : range(nombreFab_))
6     {
7         fabriquants_[i]=p.fabriquants_[i];
8     }
9 }

```



	Test	Résultat attendu	Résultat obtenu	
✓	<pre> Piece p1("roue", 0); Piece p2("retroviseur", 1); Piece p3("capot", 2); // Ajout des fabricants aux pièces p1.ajouterFab("GoodYear"); p1.ajouterFab("Michelin"); p2.ajouterFab("BMW"); p2.ajouterFab("Chevrolet"); p3.ajouterFab("BMW"); p3.ajouterFab("Chevrolet"); p3.ajouterFab("Smart"); // Créé de nouvelles pieces Piece p4(p2); Piece p5(p2); </pre>			✓
✓	<pre> Piece p6("roue", 0); Piece p7(p6); cout << (p7.getId() == p6.getId() + 1); cout << (p7.getPrix() == p6.getPrix()); cout << (p7.getNom() == p6.getNom()); cout << (p7.getNombreFab() == p6.getNombreFab()); cout << (p7.getNombreFabMax() == p6.getNombreFabMax()); </pre>	11111	11111	✓



	Test	Résultat attendu	Résultat obtenu	
✓	<pre> Piece p6("roue", 0); p6.ajouterFab("BMW"); Piece p7(p6); p6.ajouterFab("Cassy"); cout << (p7.getFabriquants() != p6.getFabriquants()); cout << (p7.getFabriquants()[0] == p6.getFabriquants()[0]); cout << (p7.getFabriquants()[1] != p6.getFabriquants()[1]); </pre>	111	111	✓

Tous les tests ont été réussis ! ✓

Solution de l'auteur de la question (Cpp):

```

1 Piece::Piece(const Piece& p) : nom_(p.nom_
2                               maxFab_(p.maxFab_)
3 {
4     fabriquants_ = new std::string[maxFab_]
5     for(auto i : range(nombreFab_))
6         fabriquants_[i] = p.fabriquants_[i]
7 }
8

```

Correct

Note pour cet envoi : 3,00/3,00.



Question 4

Correct

Note de 3,00
sur 3,00

- Modifiez le code déjà écrit afin de rendre la classe Point générique (ses attributs seront d'un type T).
- Vous devez aussi mettre la définition du constructeur à l'extérieur de la classe plutôt que dans la classe comme il l'est actuellement (cette partie de la question n'est pas testée par les tests CodeRunner).
- Ne changez pas les noms des attributs et laissez-les public.

Nous voulons qu'un code comme le suivant ait deux points dont les types des attributs sont différents (on compile en C++17):

```
Point pointInt(3, 4);  
Point pointDouble(3.5, 4.5);
```

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1  template <typename T>  
2  class Point  
3  {  
4  public:  
5      Point(T i, T j);  
6  
7      T i_;  
8      T j_;  
9  };  
10  
11  template <typename T>  
12  Point<T>::Point(T i, T j): i_(i), j_(j) {
```



	Test	Résultat attendu	Résultat obtenu	
✓	Point p(3, 4); cout << is_same_v<decltype(p.i_), int> << is_same_v<decltype(p.j_), int>;	11	11	✓
✓	Point p(3, 4); cout << (p.i_ == 3) << (p.j_ == 4);	11	11	✓
✓	Point p(3.5, 4.5); cout << is_same_v<decltype(p.i_), double> << is_same_v<decltype(p.j_), double>;	11	11	✓
✓	Point p(3.5, 4.5); cout << (p.i_ == 3.5) << (p.j_ == 4.5);	11	11	✓

Tous les tests ont été réussis ! ✓

Solution de l'auteur de la question (Cpp):

```

1  template <typename T>
2  class Point
3  {
4  public:
5      Point(T i, T j);
6
7      T i_;
8      T j_;
9  };
10
11  template <typename T>
12  Point<T>::Point(T i, T j): i_(i), j_(j) {

```

Correct

Note pour cet envoi : 3,00/3,00.



Question **5**

Terminé

Non noté

Sur mon honneur, je (écrire votre nom) , affirme que cet examen par moi-même, sans communication avec personne (autre que les enseignants indiqués en première page en cas de problème), et selon les directives identifiées dans cet énoncé d'examen.

Question **6**

Non répondue

Non noté

Si nécessaire, inscrivez vos suppositions ici, en précisant pour chaque supposition le numéro de la question concernée.

